Replay-based Strategy Prediction and Build Order Adaptation for StarCraft AI Bots

Ho-Chul Cho
Dept. of Computer Science and Engineering,
Sejong University, Seoul,
South Korea
chc2212@naver.com

Kyung-Joong Kim*
Dept. of Computer Science and Engineering,
Sejong University,
Seoul, South Korea
kimkj@sejong.ac.kr

Sung-Bae Cho
Dept. of Computer Science,
Yonsei University,
Seoul, South Korea
sbcho@cs.yonsei.ac.kr

***Abstract*—StarCraft is a real-time strategy (RTS) game and the choice of strategy has big impact on the final results of the game. For human players, the most important thing in the game is to select the strategy in the early stage of the game. Also, it is important to recognize the opponent's strategy as quickly as possible. Because of the "fog-of-war" in the game, the player should send a scouting unit to opponent's hidden territory and the player predicts the types of strategy from the partially observed information. Usually, expert players are familiar with the relationships between two build orders and they can change the current build order if his choice is not strong to the opponent's strategy. However, players in AI competitions show quite different behaviors compared to the human leagues. For example, they usually have a pre-selected build order and rarely change their order during the game. In fact, the computer players have little interest in recognizing opponent's strategy and scouting units are used in a limited manner. The reason is that the implementation of scouting behavior and the change of build order from the scouting vision is not a trivial problem. In this paper, we propose to use replays to predict the strategy of players and make decision on the change of build orders. Experimental results on the public replay files show that the proposed method predicts opponent's strategy accurately and increases the chance of winning in the game.**

Keywords—Strategy; Prediction; StarCraft; Build Order; Adaptation; Decision Tree; Feature Expansion

I. INTRODUCTION

In StarCraft, each player comes with a strategy (usually represented as a build order) given to the game map and opponents. When the game starts, each player follows the prepared build orders. At the same time, they plan to send a scouting unit to the opponent's area. Although the operation of the scouting is optional, it is nearly mandatory in human games. If the unit arrived in opponent's area successfully, it can give limited vision (around him) to the player.

There are a lot of difficulties to recognize opponent's strategy: 1) The amount of information from the scouting unit is proportional to the survival time and active movement in the enemy's territory. However, it requires a careful control of units to avoid attacks from enemy's force. 2) In the fog-of-war conditions, the vision allows the player to see only the limited area around the scouting unit. Because other areas are invisible to the player, it has high uncertainty to infer the current states of place visited or non-visited.

In case that the prediction of strategy is successful, there is risk to make decision on the change of build orders. For expert players, they have common sense on the choice of build orders when they recognize the opponent's strategy with uncertainty. Usually, the knowledge is not formalized to be used in the AI bots. The decision should be related to the accuracy of the prediction and the winning ratio of specific build orders.

Although there are some works on the design of the scouting unit control mechanism, it is still under developed to be used in the competition [1]. The scouting unit should survive in the enemy's area for long time and navigate the area continuously to update information. In addition, they need to be defensive to the attack of enemy. For human players, the controlling of scouting unit is one of the key factors to win the game.

The design of the strategy recognizer and the change of build order is not a trivial problem. It is necessary to automate the design process with the help of huge amount of data from the web. The replays on the web are important resource to learn the recognizer and the build order change. For example, Weber *et al*. run machine learning algorithms on the data extracted from replays of high-level players [2]. Although the results are promising, their extractor ignores the "fog-of-war" and the data file contains all of the opponent's information (usually not visible to the player).

In this work, we propose to use the replays to design "strategy predictor" and change "build orders." In the strategy prediction, we found that the best learning algorithms for the prediction is different on the stage of the game. Based on the observation, we combine machine learning algorithms with feature-expanded decision trees (usually perform well in the later parts of the game). From the replays, it is possible to get statistical data on the relationships among build orders. If build order A is effective on the build order B, the winning ratio from the replays could be saved to make decisions. Finally, we extract the information from the replays

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (2013-016589, 2010-0018950).

* corresponding author

978-1-4673-5311-3/13/$31.00 ©2013 IEEE

considering the "fog-of-war." It allows the extracted information should be limited to the visible to players. The realistic settings could give practical insight on the use of replays for the design of AI in the competition.

II. BACKGROUNDS

A. StarCraft Strategy

StarCraft is a popular real-time strategy game where players collect resources (minerals and gas), construct buildings and produce units to attack other players. The goal of the game is to eliminate all the buildings of opponents. However, when the player gives up the game, it stops although there are buildings remaining for the player. There are three races: Protoss, Terran, and Zerg. Each race has different units, buildings, and upgrade options.

For each race, there are a lot of different ways to determine building orders (a sequence of building construction) and unit production schedule. Because players have limited resource and time for the construction and production, it is desirable to select one strategy and optimize their actions to the choice. In fact, there is no golden strategy that beats all other strategies. Each strategy has strong and weak points to other strategies and experienced players have knowledge on the relationships among them.

It is possible to prepare strong build orders but its value is highly dependent on the opponent's strategy in the game. It is important to collect information on the enemy's territory and guess the strategy. If the prepared choice is risky in the context of opponent's choice, the player should change the current build order. The decision making on the build order is not easy because the information on the opponent is imperfect. Each player has limited vision around alliance units and the scouting unit is not guaranteed to survive for long time.

B. StarCraft AI Competition

StarCraft AI competition has been held as special events in game AI conferences (IEEE CIG and AIIDE). In the competition, each participant submits an AI program and each program plays multiple games against other entries. Because each entry has different styles of playing, it is difficult to get general goodness over the diversity of strategy. Unlike human leagues, the bots usually not change their build orders and pay small attention on the scouting.

In human games, players usually change their strategy in the next game with the same opponent. However, bots usually have no mechanism to change their build order when they lose game to the opponent. If the build order programmed is not working to the opponent, there chance to lose all the games with the player. It is not yet common to prepare multiple build orders and adaptively change (not randomly) its choice during the game. It requires "strategy prediction," "scouting" and "build order adaptation" mechanisms.

From the list of entries of the competitions, we investigate sixteen bots from their source code and replays to check the existence of scouting. Table 1 summarizes the entries show the scouting behavior (31% of the Bots). However, their main function is not observation but disturbance to the construction and production of opponents.

Table 1. Scouting in StarCraft AI Competitions

AI Players	Race	Scouting Behavior
Nova	Terran	Disturbance
Skynet	Protoss	Disturbance
UalbertaBot	Protoss	Disturbance
ItalyUndermind	Zerg	Disturbance
SPAR	Protoss	Observation

We have developed Xelnaga for the StarCraft AI competition since 2011. Like other entries in the early days, our submission has only one "build order" specialized in the use of "Dark Templar." As expected, the strategy is successful to beat players but fail to beat all entries. In the version, we have no mechanism to scout the opponent's region. The focus is to follow the predefined build order until we have enough attack units. This is a kind of all-or-nothing strategy. It is weak to the very early attack or "observer" production strategy.

In 2012, we add a function to scout the opponent's are using probe (a resource collecting unit in PROTOSS). From the observation, we count the assimilator and the gateway in the enemy's territory. If the number of gateway (attack unit production building) is more than three and no assimilator (gas extractor), we recognize it as "very fast attack". If the number of gateway is two, the strategy is "fast attack". Although we use very simple rules, they're helpful to recognize the early attack and change our strategy. However, it is a still big problem to recognize a variety of strategies and handle them properly by changing our build orders. However, it is a still big problem to recognize a variety of strategies and handle them properly by changing our build orders.

C. StarCraft Replay Mining

Game mining is an interdisciplinary research to extract useful knowledge from game-related data sources with data mining techniques [3]. The knowledge can be used to build better games and artificial intelligence for the games [4][5]. There are several sources of information originated from games: transcripts, logs, text chats, social networks, and so on. It can be used to mine the behavior of gamers to improve the design of the games [6][7][8].

Since gamers have played the StarCraft for more than ten years, their replays have been archived in game-related portals. AI researchers analyze the replays to build a "strategy" prediction model using CBR [9], J48, k-NN, NNge [2] and Bayesian network [10]. Also, it helps to find the goal of players [11] and relationships among build-orders (strong or weak) [12].

Because the previous works are dependent on some replay programs without "fog-of-war" options, they pose unrealistic assumption that players have full vision to the opponent. Recently Park *et al.* tried to predict that opponent's strategy with fog-of-war [1]. In the experiments, they use their agents

(bots) to records realistic (with fog-of-war) observation during the game. Although it is successful to collect realistic logs but it should play the games against other agents to get the data. So, the approach is not useful to analyze the replays. In our 2013 version, we develop a technique to collect realistic logs from replays with an "observer" role agent. The software simulates each game from replay in a fastest mode and the observer records all the relevant game events considering the "fog-of-war." Hostetler *et al*. infer the strategy in fog-of-war [13]. Their focus is to infer the current opponent's unobserved information from observed data.

III. Proposed Methods

In this paper, we propose a framework to exploit the replays to predict strategy of opponent and making decision on the change of build orders. Fig. 1 shows the overview of the proposed framework.

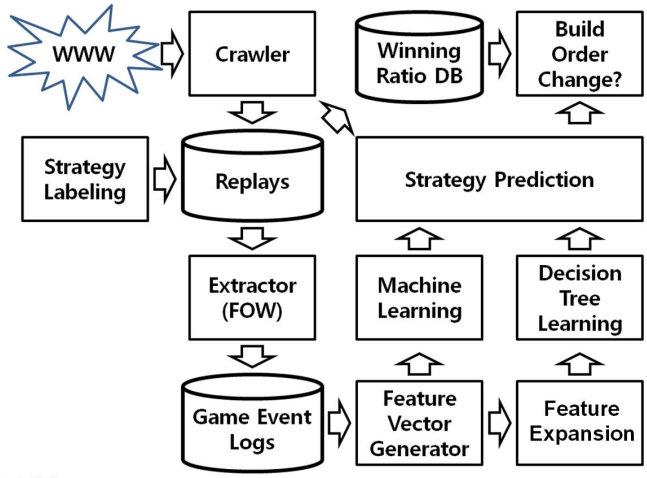


Fig. 1. Overview of the proposed method (FOW = Fog of War)

The replays are available from famous internet game portals. However, there is no information on the strategy used in the replay. The strategy "labeling" can be done by human experts. The labeled replays can be used as a training data for the supervised learning in the next step. It is necessary to automate the "labeling" by modeling the human experts. The replays record all the gaming events (mouse click, the production of units, the construction of buildings and upgrades) in a binary format.

The Extractor converts the raw replay files into human readable text files. Because the Blizzard, the creator of the StarCraft, does not support the conversion, the extraction is dependent on some software personally developed by experts. They're Lord Martin Replay Browser and BWChart. Because the extraction software is built without support from the game creator, it has several limitations. For example, it has limited support on the multiplayer games and no option to reflect the "fog-of-war" in the games.

The next step is to build a feature vector from the raw text files. Because the replays store all the events necessary to recover the games again, it has large amount of useless information to predict the "strategy." In this step, expert knowledge is required to the choice of features. Although it records all the information on the units, buildings and user commands but most of them are not useful. Simply, it is possible to set filters to delete useless information. Because the raw data is too coarse, additional preprocessing techniques (averaging, counting and so on) can be applied.

In previous works, machine learning algorithms are successful to predict the strategy of opponents in the early stage of the games [1][2]. However, they're not interpretable to human experts who design the build orders of the bots. Also, it is not easy to maintain high accuracy throughout the games (early, middle and end stages). As a solution, we propose to use a decision tree to predict the strategy. Because the model is interpretable to human experts, it is straightforward to convert them into a build order. To enhance the performance of decision tree, we expand a feature set for the model by incorporating new features based on time comparison. Because single machine learning algorithm fails to cover all the stage of the games, we propose to assign different machine learning models for each stage.

Based on the prediction, the player should make decision on the change of the build order. There are big uncertainties in the decision. In this architecture, we automatically get the statistics on the winning ratio of strategy against others. For example, it stores the winning ratio if strategy *A* plays against strategy *B*. The final decision is based on the prediction accuracy and the winning ratio for the predicted strategy and the player's current strategy.

A. Replay Preprocessing

Although replays are stored in a binary format, it is possible to convert them to game logs using Lord Martin Replay Browser[2] or BWAPI[3]. They extract types of buildings, units, upgrade and their making time from replays. The extracted raw data is encoded as a feature vector, containing temporal features. If units or buildings are produced or constructed multiple times during the game, each feature describes the time when they're made first. For example, the PROTOSS player constructs multiple Gateways (buildings for the attack unit production) during the game. But in the feature vector, the feature "Gateway" stores only the time that the first Gateway is constructed.

$$f(x)_P = \begin{cases} t & \text{time when } x \text{ is first produced by } P \\ 0 & x \text{ was not (yet) produced by } P \end{cases}$$

, where x is a unit type, build type or unit upgrade. A subset of an example feature vector for a Terran player is shown in Table 2. In the game, second gas was not yet produced by the player.

Table 2. A subset of an example feature vector (from a Terran player's feature vector for a Protoss vs. Terran match)

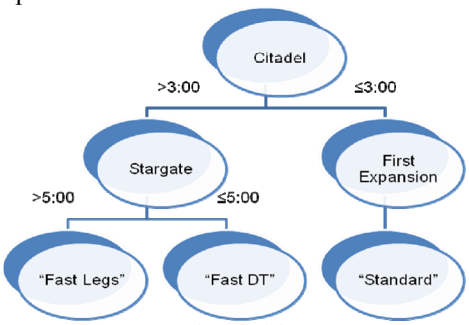
Attribute	Game Time
Pylon	1:20

[2] http://lmrb.net
[3] http://code.google.com/p/bwapi/

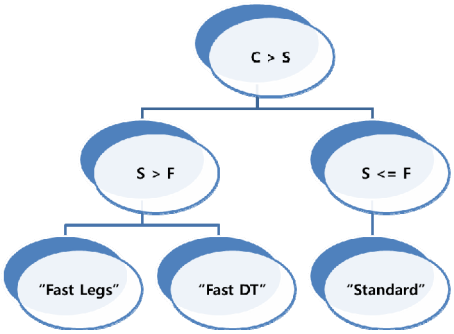
Gateway	2:05
Gas	2:40
Expansion	11:00
Second Expansion	15:11
Third Expansion	18:45
Fourth Expansion	0:00
Second Gas	0:00

B. *Strategy Prediction*

In the strategy prediction, we propose to use feature expanded decision tree. The only difference with the standard decision tree is that it incorporates a lot of new features into the original vector. In StarCraft, the order of game events (action *A* prior to action *B*) is one of important factors to identify the strategy. Fig. 2 shows an example of feature-expanded decision tree for the StarCraft.



(a) Standard decision tree



(b) Feature-expanded decision tree

Fig. 2. An example of standard and feature-expanded decision tree (C, S and F stand for "Citadel," "Stargate," and "First Expansion," respectively)

The number of features in the vector is *N*. There are $0.5 \times N \times (N-1)$ comparisons among the features (only, '>' operation is considered). The new feature has one of "true" and "false" value. Because the number of new features is large, feature selection is adopted. The percentage of "true" value for each strategy (class) is calculated. For example, x1>x2 is true for all the replays labeled as "Fast DT" strategy. The comparison is worth to be considered. If a feature shows 100% for at least one strategy (class), it is selected.

	X1>X2	X1>X3	....	XN-1>XN	Strategy
Replay1	True	False		True	Fast Legs
Replay2	False	False		True	Fast DT
....					Fast Expand
ReplayM	True	True		True	Unknown

(a) Feature expansion

	X1>X2	X1>X3	....	XN-1>XN
Fast Legs	85%	75%		30%
Fast DT	100%	90%		20%
Fast Expand	20%	80%		10%
Fast Air	50%	30%		30%
Reaver	55%	20%		60%
Standard	35%	60%		100%
Unknown	20%	75%		80%

(b) Feature selection

Fig. 3. Feature expansion and the selection for the decision tree

Also, we propose to use an ensemble approach where different machine learning models take charge of different stage of games. The separation of game stage is done by experts. Usually, the game can be divided into early, middle and end stages. In case that the expert knowledge is not available, it is possible to assign machine learning models that perform the best on the training samples at the given time. For example, we can use "Random Committee" models from the game start to 9 minutes and "Feature-Expanded DT" after the time.

C. *Build Order Change*

From the replays, it is possible to get statistics on the relatinoships among strategies. In sum, the winning ratio when strategy *A* plays against strategy *B*. From the training samples, it is possible to get prediction accuracy (0~1) on the trained models. α is the maximum winning ratio if the player changes the current build order into new one (from the statistics). β is 0.5.

$$\mathrm{E}[\mathrm{Win}] = Accuracy_{\substack{Ensemble \\ Approach}} \times \alpha + (1 - Accuracy_{\substack{Ensemble \\ Approach}}) \times \beta$$

IV. Experimental Results

A. *Experimental Setup*

In this paper, we collect StarCraft replays from YGOSU.com. The number of replays is 570 and all the games are PROTOSS vs. PROTOSS. Because we can extract text logs in the perspective of each player of the game, the number of samples is 1140. Also, we repeat the extraction two times by controlling the "fog-of-war" options. As a result, we have two sets of data samples ("with fog-of-war" and "without fog-of-war"). The number of features in the vector is 56.

Also, we use the data[4] from Weber *et al*. [2]. Because they already preprocessed the raw replay files, it is easy to use for the experiments. Also, they have data for games among all races (PvP, PvT, PvZ, TvT, TvZ, and ZvZ). For example, PvP represents PROTOSS versus PROTOSS. The number of samples for each type of games is ranging from 542 to 1150. In total, they have 5493 samples. However, they do not consider the "fog-of-war" in the log extraction. Also, the raw replay files are not available for the data. It makes difficult to know the player who wins the game for the replay. The number of features for PROTOSS, TERRAN, and ZERG is 56, 51, and 48, respectively.

Table 3. The number of samples (FOW = Fog-of-War) (P = PROTOSS, T = TERRAN, Z = ZERG)

	Types	FOW	Raw Replays	# Samples
YGOSU.com	P vs. P	O	O	1140
	P vs. P	-	O	1140
Weber *et al*.	P vs. P	-	-	542
	P vs. T	-	-	1139
	P vs. Z	-	-	1024
	T vs. T	-	-	628
	T vs. Z	-	-	1150
	Z vs. Z	-	-	1010

Table 3 summarizes the details of data used. The number of strategies for each race is seven. For example, the PROTOSS has "Fast Dark Templar," "Fast Observer," "Fast Expansion," "Fast Legs," "Reaver Drop," "Carrier," and "Unknown." Ten-fold cross validation is used for all experiments. The machine learning algorithms are implemented with the WEKA API [14]. The machine learning algorithms are evaluated at different time steps throughout the game. The overall performance is defined as the average accuracy during the game. N is the number of sampling points during the game (in this paper, N=31, Game Time = 15 min).

$$Avg\_Acc = \frac{1}{N}\sum_{t=0}^{GameTime} Accuracy_{Classifier}(t)$$

B. Feature-Expanded Decision Tree

Table 4. Comparison of standard DT and the feature-expanded DT in terms of accuracy and the size of model (the number of leaves and the size of the tree) (W =Weber dataset, Y=YGOSU.com)

Race (Source)	Standard DT		Feature-Expanded DT	
	Accuracy (%)	Size	Accuracy (%)	Size
P (Y)	89.49	(157, 313)	99.73	(15, 29)
P (W)	89.68	(125, 249)	99.77	(14, 27)
T (W)	91.05	(122, 243)	99.96	(11, 21)
Z (W)	95.76	(72 , 143)	100.0	(10, 19)
Average	91.50	(119, 237)	99.87	(13, 24)

[4] http://eis.ucsc.edu/StarCraft_Data_Mining

The purpose of the "feature-expanded" decision tree (FBDT) is to build machine learning models interpretable to humans and easily converted into programming codes (as a build order categorization). The algorithm is applied to the replays from each race. For comparison, the standard DT (without feature expansion) is used. It shows that the FBDT is accurate compared to the standard DT and the size of the model is relatively small. Also, the result from our YGOSU.com data is similar to the Weber's dataset. Fig. 4 shows an example of conversion from the FBDT into a programming code.

```
IF (FirstExpansion <= Stargate){
   IF(RoboBay <= FirstExpansion){
      IF(Citadel <= RoboBay){
         IF(Legs <= Archives){
            IF(FourthExpansion <= Legs) "Unknown"
            ELSE "Fast Legs"
         }
         ELSE "Fast DT"
      }
      ELSE{
         IF(RoboSupport <= Observory){
            IF(SecondExpansion <= RoboSupport) "Unknown"
            ELSE "Reaver Drop"
         }
         ELSE "FastObs"
      }
   }
   ELSE{
      IF(FirstExpansion <= Citadel) "Fast Expand"
      ELSE{
         IF(Legs <= Archives) "Fast Legs"
         ELSE "Fast DT"
      }
   }
}
ELSE{
   IF(Citadel <= Stargate){
      IF(Legs <= Archives) "Fast Legs"
      ELSE "Fast DT"
   }
   ELSE{
      IF(RoboBay <= Stargate){
         IF(RoboSupport <= FirstExpansion) "Reaver Drop"
         ELSE "Fast Obs"
      }
      ELSE "Carrier"
   }
}
```

Fig. 4. Conversion of feature-expanded decision tree into a programming code

C. "Strategy Prediction" during the Game

Table 5 summarizes the prediction accuracy of machine learning algorithms on PROTOSS vs. PROTOSS games. It shows that the results from Weber dataset are similar to the one from YGOSU.com. As expected, the introduction of "fog-of-war" decreases the prediction accuracy. It is interesting that the FBDT outperforms other classifiers in the later parts of the games. However, it is very poor in the early

stage of the games. Other machine learning algorithms perform well in the early stage of the game but not the best in the later part of the game. From this observation, it is meaningful to use more than one classifier during the game. For example, Rotation Forest is used in the early stage of the game but the FBDT in the later part of the game (Fig. 5).

Table 5. The comparison of "strategy prediction" accuracy (bold means the best accuracy)

(a) P vs. P (Weber Data)

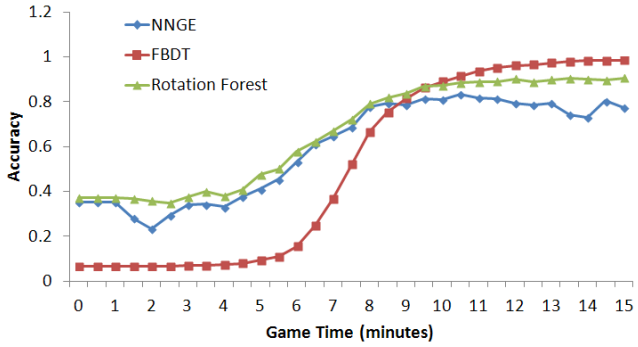
	5 min	10 min	15 min	Avg_Acc
NNGE	49.6	80.6	76.0	60.3
KNN	47.4	80.4	74.5	60.1
J48 [18]	43.9	81.9	85.4	62.1
FBDT	26.0	**89.7**	**99.6**	59.7
Bagging [17]	50.0	85.2	86.9	63.1
Random Committee	50.2	85.1	85.0	64.0
Random Forest [16]	49.8	83.2	83.4	63.7
Rotation Forest [15]	**50.4**	85.8	89.1	**65.3**

(b) P vs. P (YGOSU.com)

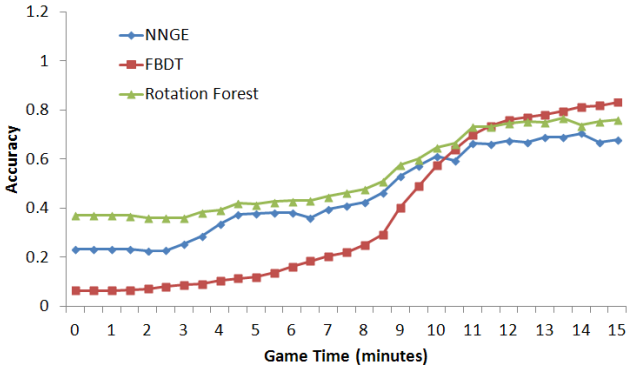
	5 min	10 min	15 min	Avg_Acc
NNGE	41.1	81.0	77.5	59.6
KNN	40.4	78.3	67.6	56.0
J48	38.6	85.5	85.0	62.8
FBDT	9.4	**89.1**	**98.5**	51.0
Bagging	**47.9**	87.2	89.4	65.7
Random Committee	44.6	84.7	86.0	63.7
Random Forest	45.4	84.7	86.3	63.5
Rotation Forest	47.5	87.4	90.6	**66.0**

(c) P vs. P (YGOSU.com) (with fog-of-war)

	5 min	10 min	15 min	Avg_Acc
NNGE	38.0	61.3	68.0	46.0
KNN	35.3	57.0	58.8	43.0
J48	37.2	61.5	72.5	50.2
FBDT	11.9	57.4	**83.2**	37.1
Bagging	40.9	63.9	75.3	53.0
Random Committee	39.5	62.5	70.1	51.4
Random Forest	40.2	61.6	70.7	51.3
Rotation Forest	**41.6**	**64.8**	76.0	**53.5**



(a) P vs. P (YGOSU.com)



(b) P vs. P (YGOSU.com) (with fog-of-war)

Fig. 5. The introduction of "fog-of-war" and the prediction accuracy during the game

D. Build Order Change

We analyze the 570 replays from YGOSU.com. It shows that the number of replays categorized into "Fast Legs" and "Carrier" is too small compared to other strategies. In the calculation of the winning ratio, we only consider the five strategies except the low-percentage strategies. Table 6 summarizes the winning ratio from the replays. It means that the Fast DT strategy wins 59% against the Fast Observer strategy.

Table 6. Winning ratio from the replay files (YGOSU.com)

(a) The number replays for each strategy in YGOSU.com data

Fast Legs	Fast DT	Fast Obs	Reaver Drop	Carrier	Fast Expand	Unknown
15	162	424	200	0	265	74

(b) Winning ratio of each strategy (0~1)

Player	Opponent				
	Fast DT	Fast Obs	Reaver Drop	Fast Expand	Unknown
FastDT	0.50	0.59	**0.67**	0.64	0.00
FastObs	0.41	0.50	**0.52**	0.49	0.29
Reaver Drop	0.33	0.48	0.50	0.52	**0.71**
Fast Expand	0.36	**0.51**	0.48	0.50	0.31
Unknown	**1.00**	0.71	0.29	0.69	0.50

Fig. 6 shows the change of the E[Win] during the game. The value is calculated using the prediction accuracy of the combined models (Random Forest and FBDT) and the winning ratio in Table 6. In the early stage of the game, the prediction accuracy is not high and it is not beneficial to change the build order. In 6~7 minutes of the game, the prediction accuracy is relatively high and the E[win] becomes the maximum. After the time, the prediction accuracy increases but the possibility of the build order change goes down "(almost buildings are constructed)". The player can make a decision on the change of build orders based on the E[win] during the game.

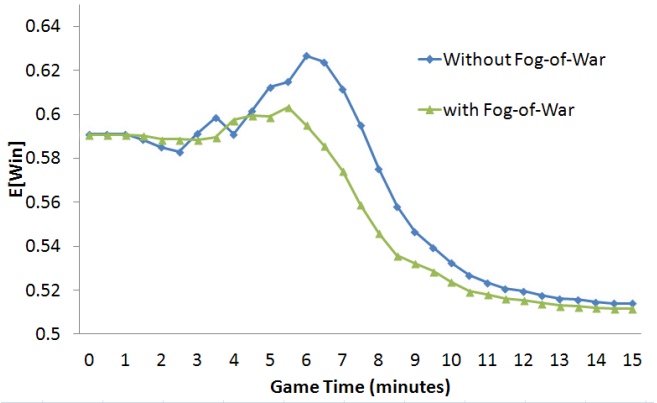


Fig. 6. Expected win (0~1) from the prediction accuracy (Rotation Forest + FBDT) and the winning ratio of each strategy (P vs. P, YGOSU.com data)

V. Conclusions and Future Works

In this paper, we propose a framework to use the replays on the automatic design of strategy prediction and the build order adaptation. For the replay mining, we develop a new customized tool for the extraction of information from the replays considering the "Fog-of-War." For the strategy prediction, we propose to use the "feature expansion" for the decision tree learning. It returns human-interpretable accurate models to predict the strategy of the game. Because the model is not good in the early stage, it is desirable to hybrid it with other machine learning algorithms (for example, rotation forest). For the build order change, we propose an equation to get the E[Win] values from the prediction accuracy and the winning ratio from the replays.

Experimental results show that the proposed FBDT is promising to build a small-size human-interpretable tree models. However, its accuracy is not good in the early stage of the game. The rotation forest is successful to predict the strategy in the early stage of the game. The combination of the two models outperforms other candidates in the strategy prediction problems. The introduction of the "Fog-of-War" in the game reduces the prediction accuracy as expected. But there is a learning algorithm robust to the uncertainty. The build order change experiments show that the 6~7 minutes are the best timing to change the build order.

Although we can build strategy prediction models with the "Fog-of-War" from the human replay files, there is difference between human and bots games. In the human replays, the players control the "scouting unit" effectively and acquire useful information to predict the strategy. However, in the bots, it is still under development to implement the "scouting unit" management. The success of the strategy prediction is highly dependent on the use of the "scouting unit" in the presence of the "Fog-of-War."

References

[1] H.-S. Park, H.-C. Cho, K.-Y. Lee, and K.-J. Kim, "Prediction of early stage opponents strategy for StarCraft AI using scouting and machine learning," *In Proceedings of the Workshop at SIGGRAPH Asia (WASA 2012)*, pp. 7-12, 2012.

[2] B. Weber, and M. Mateas, "A data mining approach to strategy prediction," *IEEE Symposium on Computational Intelligence and Games*, pp.140-147, 2009.

[3] A. Tveit, "Game usage mining: Information gathering for knowledge discovery in massive multiplayer games," *Proceedings of the International Conference on Internet Computing*, pp. 636-642, 2002.

[4] D. Kennerly, "Better game design through data mining," *Gamasutra*, 2003.

[5] K.S.Y. Chiu, and K.C.C. Chan, "Game engine design using data mining," *Proceedings of the 26th IASTED International Conference on Artificial Intelligence and Applications*, pp. 352-357, 2008.

[6] C. Thurau, and C. Bauckhage, "Analyzing the evolution of social groups in World of Warcraft," *IEEE Conference on Computational Intelligence and Games*, pp. 170-177, 2010.

[7] T. Mahlmann, A. Drachen, J. Togelius, A. Canossa, and G. N. Yanakakis, "Predicting player behavior in Tomb Raider: Underworld," *IEEE Conference on Computational Intelligence and Games*, pp. 178-185, 2010.

[8] K.-J. Shim, and J. Srivastava, "Behavioral profiles of character types in EverQuest II," *IEEE Conference on Computational Intelligence and Games*, pp. 186-194, 2010.

[9] J.-L. Hsieh, and C.-T. Sun, "Building a player strategy model by analyzing replays of real-time strategy games," *IEEE International Joint Conference on Neural Networks*, pp. 3106-3111, 2008.

[10] G. Synnaeve and P. Bessiere, "A Bayesian model for opening prediction in RTS games with application to StarCraft," *in Proceedings of 2011 IEEE CIG*, Seoul, South Korea, pp. 281-288, Sep. 2011.

[11] B. Weber, and S. Ontanon, "Using automated replay annotation for case-based planning in games," *International Conference on Case-based Reasoning Workshop on CBR for Computer Games*, pp. 15-24, 2010.

[12] J. K. Kim, K.-H. Yoon, T. Yoon, and J.-H. Lee, "Cooperative learning by replay files in real-time strategy game," *Lecture Notes in Computer Science*, vol. 6240, pp. 47-51, 2010.

[13] J. Hostetler, E. W. Dereszynski, T. G. Dietterich, and A. Fern, "Inferring Strategies from Limited Reconnaissance in Real-time Strategy Games," *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence (UAI2012)*, pp. 367-376, 2012.

[14] I. H. Witten, E. Frank and M. A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann, 2011.

[15] J. J. Rodriguez, L. I. Kuncheva, and C. J. Alonso, "Rotation forest: A new classifier ensemble method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 10, pp. 1619-1630, 2006.

[16] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5-32, 2001.

[17] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123-140, 1996.

[18] R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.